

Online scheduling of bounded length jobs to maximize throughput

Christoph Dürr · Łukasz Jeż · Nguyen Kim Thang

Published online: 9 April 2011

© The Author(s) 2011. This article is published with open access at Springerlink.com

Abstract We consider an online scheduling problem, motivated by the issues present at the joints of networks using ATM and TCP/IP. Namely, IP packets have to be broken down into small ATM cells and sent out before their deadlines, but cells corresponding to different packets can be interwoven. More formally, we consider the online scheduling problem with preemptions, where each job j is revealed at release time r_j , and has processing time p_j , deadline d_j , and weight w_j . A preempted job can be resumed at any time. The goal is to maximize the total weight of all jobs completed on time. Our main results are as follows. Firstly, we prove that when the processing times of all jobs are at most k , the optimum deterministic competitive ratio is $\Theta(k/\log k)$. Secondly, we give a deterministic algorithm with competitive ratio depending on the ratio between the smallest and the largest processing time of all jobs. In particular, it attains competitive ratio 5 in the case when all jobs have *identical* processing times, for which we give a lower bound of 2.598. The latter upper bound also yields an $O(\log k)$ -competitive randomized algorithm for the variant with processing times bounded by k .

Keywords Online scheduling · Preemption with resume · Competitive analysis · Online algorithms

1 Introduction

Many Internet service providers use an ATM network which has been designed to send telephone communication and television broadcasts, as well as usual network data. However, the Internet happens to use TCP/IP, so at the joints of these networks IP packets have to be broken down into small ATM cells and fed into the ATM network. This raises many interesting questions, as an ATM network works with fixed sized cells (48 bytes), while an IP network works with variable sized packets. In general, packet sizes are bounded by the capacity of Ethernet, i.e., 1500 bytes, and in many cases they actually achieve this maximal length. Ideally, packets also have deadlines and priorities (weights). The goal is to maximize the *quality of service*, i.e., the total weight of packets that have been entirely sent out on time.

This problem can be formulated as an online scheduling problem on a single machine, where jobs arrive online at their release times, have some processing times, deadlines, and weights, and the objective is to maximize the total weight of jobs completed on time. Preemption is allowed, so a job i can be scheduled in several separated time intervals, as long as their lengths add up to p_i . Time is divided into integer time steps, corresponding to the transmission time of an ATM cell, and all release times, deadlines, and processing times are assumed to be integer. The algorithm selects the job to schedule at the very beginning of the step, and cannot switch to another job until the step is over. This problem can be denoted as $1|\text{online-}r_i; \text{pmtn}| \sum w_i(1 - U_i)$, according to the notation of Chen et al. (1998).

Preliminary version appeared in *Proceedings of the 7th International Workshop on Approximation and Online Algorithms*. Work was conducted while C.D. and N.K.T. were affiliated at LIX, Ecole Polytechnique, Palaiseau, France.

C. Dürr
CNRS, LIP6, Université Pierre et Marie Curie, Paris, France

Ł. Jeż (✉)
Institute of Computer Science, University of Wrocław,
50-383 Wrocław, Poland
e-mail: lje@cs.uni.wroc.pl

N.K. Thang
Lamsade, Université Paris-Dauphine, Paris, France

1.1 Our results

In this paper we consider the case when processing times of all jobs are bounded by some constant k , and the case when they (approximately) equal k . Both variants are motivated by the network application in mind. We study the competitive ratio as a function of k . Our main results are as follows.

- We provide an optimal deterministic online algorithm for the bounded processing time case that reaches the ratio $O(k/\log k)$. We also develop an $O(\log k)$ -competitive randomized algorithm (in the oblivious adversary model), based on our next result and the “classify and randomly select” paradigm (Awerbuch et al. 1994).
- We give an online deterministic algorithm whose competitive ratio depends on the ratio between the minimum and maximum processing times of a job. In particular, it is $O(1)$ -competitive when the latter ratio is constant. For the special case of identical processing times (ratio 1), the algorithm is 5-competitive. We also prove a lower bound of $3\sqrt{3}/2 \approx 2.598$ on the competitive ratio of any deterministic algorithm for that special case, slightly improving a bound of 2.59 due to Chan et al. (2004). In fact, we give a more involved analysis of their construction, pinning down the threshold competitive ratio that it can force.

We also provide several minor results for completeness of the paper.

- For the bounded processing time case, we show that the competitive ratio of a well-known SMITH RATIO ALGORITHM is between k and $2k$. We also show that asymptotically the competitive ratio of any deterministic algorithm is at least $k/\ln k$, improving the previous bound of $k/(2\ln k) - 1$ (Ting 2008) by a factor of 2.
- For bounded processing time with unit weights, the competitive ratio is $\Omega(\log k/\log \log k)$ when time points are allowed to be rationals (Baruah et al. 1994). We provide an alternative proof for the more restricted integer variant.
- $O(\log k)$ -competitiveness of the SHORTEST REMAINING PROCESSING TIME FIRST algorithm for the bounded processing time, unit weight model follows as a byproduct from an involved analysis of Kalyanasundaram and Pruhs (2003). We provide an alternative concise proof of its $2H_k$ -competitiveness and note that this is tight up to a constant factor.

1.2 Related work

It is known that the general problem without a bound on processing times has an unbounded deterministic competitive ratio, even if all jobs have unit weights (Baruah et al. 1994), so different directions of research were considered. Two related approaches are to consider resource augmentation and

randomization. For the former there is an online algorithm that has constant competitive ratio provided it is allowed a constant speed-up of its machine compared to the adversary (Kalyanasundaram and Pruhs 2000). For the latter a constant competitive randomized algorithm is known for the unit weights of jobs variant (Kalyanasundaram and Pruhs 2003). Finally, a third direction is to consider instances with restrictions on the processing times.

Bounded processing time, unit weights ($\forall j \ p_j \leq k, w_j = 1$) The offline problem can be solved in time $O(n^4)$ (Baptiste 1999) even when the processing time is unbounded. Baruah et al. (1994) showed that any deterministic online algorithm is $\Omega(\log k/\log \log k)$ -competitive in a model where processing times, release times, and deadlines of jobs can be rational. The currently best known algorithm is SHORTEST REMAINING PROCESSING TIME FIRST, which is $O(\log k)$ -competitive (Kalyanasundaram and Pruhs 2003). The same paper provides a constant competitive randomized algorithm, however with a large constant.

Bounded processing time, arbitrary weights ($\forall j \ p_j \leq k$) For fixed k the offline problem has not been studied to our knowledge, and when the processing times are unbounded, the offline problem is \mathcal{NP} -hard by a trivial reduction from the Knapsack Problem. It is known that any deterministic online algorithm for this case has competitive ratio $k/(2\ln k) - 1$ (Ting 2008). For the variant with tight jobs only, i.e., jobs that satisfy $d_j = r_j + p_j$, Canetti and Irani (1998) provide an $O(\log k)$ -competitive randomized online algorithm and show a $\Omega(\sqrt{\log k/\log \log k})$ lower bound for any randomized competitive algorithm against an oblivious adversary.

Equal processing time, unit weights ($\forall j \ p_j = k, w_j = 1$) The offline problem can be solved in time $O(n \log n)$ (Lawler 1994), and the solution can be turned into a 1-competitive online algorithm, see for example (Vakhania 2008).

Equal processing time, arbitrary weights ($\forall j \ p_j = k$) The offline problem can be solved in time $O(n^4)$ (Baptiste et al. 2004). For $k = 1$ the problem is well studied, and the deterministic competitive ratio is between 1.618 and 1.83 (Hajek 2001; Englert and Westermann 2007).

Our model is sometimes called the *preemptive model with resume*, as opposed to *preemptive model with restarts* (Chrobak et al. 2007), in which an interrupted job can only be processed from the very beginning. *Overloaded real-time systems* (Baruah et al. 1994) form another related model, in which all the job parameters are reals, the time is continuous, and uniform weights are assumed.

2 Preliminaries

For a job i we denote its release time by r_i , its deadline by d_i , its processing time by p_i and its weight by w_i . All these quantities, except w_i , are integers. Let $q_i(t)$ be the remaining processing time of job i for the algorithm at time t . When there is no confusion, we simply write q_i . We say that job i is *pending* for the algorithm at time t if it has not been completed yet, $r_i \leq t$, and $t + q_i(t) \leq d_i$. We say a job j is *tight* at time t if $t + q_j(t) = d_j$. For a job j uncompleted by the algorithm, the *critical time* of j is the latest time when j was still pending for the algorithm. In other words, the critical time s of job j for the algorithm is a moment s such that if the algorithm does not schedule j at time s , it cannot finish j any more, i.e., $s = \max\{\tau : \tau + q_j(\tau) = d_j\}$.

As time is divided into steps, every job i naturally consists of p_i units. We denote these units by (i, a) , for $1 \leq a \leq p_j$, where (i, a) stands for the unit of i whose processing started when there were a units remaining, i.e., when $q_i = a$. We assume that a unit (i, a) scheduled at time t is processed during the time interval $[t, t + 1)$, i.e., its processing is finished just before time $t + 1$. For this reason by *completion time* of a job i we mean $t + 1$ rather than t , where t is the time its last unit was scheduled.

Throughout the paper we analyze many algorithms with similar charging schemes sharing the following outline: for every job j completed by the adversary we consider its p_j units. Each unit of job j charges w_j/p_j to some job i_0 completed by the algorithm. The charging schemes satisfy the property that every job i_0 completed by the algorithm receives a total charge of at most Rw_{i_0} , which implies R -competitiveness of the algorithm.

More precisely we distinguish individual units scheduled by both the algorithm and the adversary. With every algorithm's unit (i, a) we associate a *capacity* $\pi(i, a)$ that depends on w_i and a , whose exact value will be different from proof to proof. The algorithms, with their capacities, will be designed in such a way that they satisfy the following properties, with respect to π .

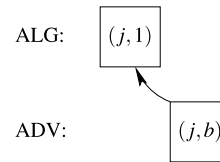
ρ -monotonicity: If the algorithm schedules (i, a) with $a > 1$ at t and (i', a') at $t + 1$, then $\rho\pi(i', a') \geq \pi(i, a)$,

validity: If a job j is pending for the algorithm at any time t , then the algorithm schedules a unit (i, a) at t such that $\pi(i, a) \geq w_j/p_j$.

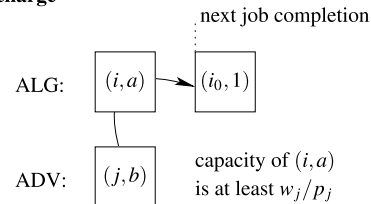
Let us remark that our algorithms are ρ -monotone for some positive $\rho < 1$. We note the following.

Observation 1 *A valid algorithm schedules some job whenever there is one pending. If the algorithm is also ρ -monotone for some positive ρ , it has no idle time between scheduling the job and completing some, possibly other, job.*

type 1 charge



type 2 charge



type 3 charge

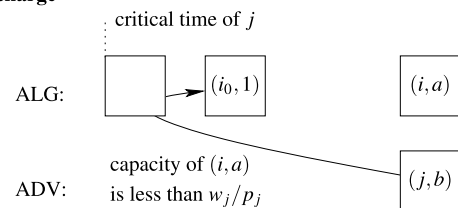


Fig. 1 The general charging scheme

We distinguish three types of charges in the charging scheme; these are depicted in Fig. 1. Let (j, b) be a unit of job j scheduled by the adversary at time t .

Type 1: If the algorithm already completed j by time t , then charge w_j/p_j to j .

Type 2: Otherwise if the algorithm schedules a job unit (i, a) at time t that has capacity at least w_j/p_j , then we charge w_j/p_j to i_0 , where i_0 is the next job completed by the algorithm from time $t + 1$ on.

Type 3: In the remaining case, j is not pending anymore for the algorithm by its validity. Let s be the critical time of j . We charge w_j/p_j to i_0 , where i_0 is the first job completed by the algorithm from time $s + 1$ on.

Clearly every job i_0 completed by the algorithm can get at most p_{i_0} charges of type 1 summing up to at most w_{i_0} . We can bound the other types as well.

Lemma 1 *Let $0 < \rho < 1$. Then the total type 2 charge a job i_0 completed by a ρ -monotone and valid algorithm receives is at most $\pi(i_0, 1)/(1 - \rho)$.*

Proof Let t_0 be the completion time of i_0 , and let s be the smallest time such that $[s, t_0)$ contains no idle time and no other job completion. Then the unit scheduled at time $t_0 - i$ for $1 \leq i \leq t_0 - s + 1$ has capacity at most $\pi(i_0, 1)\rho^{i-1}$,

by ρ -monotonicity. Type 2 charges to i_0 can only originate from units scheduled by the adversary in $[s, t_0]$: those from steps after t_0 , if any, are charged to jobs completed by the algorithm later, and those from steps before s , if any, are charged to jobs completed by the algorithm before by Observation 1. Thus the total type 2 charge to i_0 is bounded by

$$\pi(i_0, 1)(1 + \rho + \rho^2 + \rho^3 + \cdots) = \pi(i_0, 1)/(1 - \rho). \quad \square$$

Lemma 2 *Let \mathcal{J} be the set of job units that are type 3 charged to a job i_0 completed by a monotone and valid algorithm. Then for all p there are at most $p - 1$ units $(j, b) \in \mathcal{J}$ s.t. $p_j \leq p$. In particular, $|\mathcal{J}| \leq k - 1$ if all jobs have processing time at most k . Moreover, $w_j/p_j \leq \pi(i_0, 1)$ holds for each $(j, b) \in \mathcal{J}$.*

Proof To be more precise we denote the elements of \mathcal{J} by triplets (s, t, j) such that a job unit (j, b) scheduled at time t by the adversary is type 3 charged to i_0 and its critical time is s . Let $t_0 \geq s$ be the completion time of i_0 by the algorithm. As j was pending at time s , and between s and t_0 there is no idle time, nor any other job completion, by monotonicity and validity of the algorithm the capacities of all units in $[s, t_0]$ are at least w_j/p_j . In particular, $\pi(i_0, 1) \geq w_j/p_j$.

By definition of type 3 charges, the algorithm schedules some unit with capacity strictly smaller than w_j/p_j at t , so $t_0 \leq t$.

Since s is the critical time of j , $s + q_j(s) = d_j$. However, since the adversary schedules j at time t we have $t < d_j$. Thus $t - s < q_j(s) \leq p_j$. Note that all triplets $(s, t, j) \in \mathcal{J}$ have distinct times t . Note that there can be at most $c - 1$ pairs (s, t) with distinct t that satisfy $s \leq t_0 \leq t$ and $t - s < c$. \square

In the following sections, we adapt this charging scheme to individual algorithms, demonstrating that the class of algorithms that can be analyzed this way is quite rich. However, while in each case we consider an algorithm that is monotone and valid with respect to an appropriate capacity function, and classify charges as described, to obtain good bounds we often tweak the scheme in some way. Specifically, on one occasion (Proposition 1) we improve upon the bound given by Lemma 1 by further analysis, and on another (Theorem 2) we redesign the whole scheme so that there are no charges of type 3. We also note that even with the analysis framework fixed and without any tweaks one still needs to design the algorithm carefully, and then appropriately choose the capacity function. In particular, it is possible to analyze a fixed algorithm using different capacity functions, and their choice greatly affects the upper bound on the algorithm's competitive ratio one obtains.

All our algorithms at every step schedule the job with maximum capacity, but this is not required for the scheme to

work. For example, some of our preliminary algorithms did not work this way. Note that our algorithms need to select jobs only at release times or completion times of some jobs.

3 Bounded processing time, unit weights

In this section we consider instances in which every job has processing time at most k and unit weight, i.e., $w_i = 1$ for all jobs i .

3.1 Upper bound

THE SHORTEST REMAINING PROCESSING TIME FIRST Algorithm is a greedy online algorithm that schedules at every step the pending job with the smallest remaining processing time.

It was analyzed in Kalyanasundaram and Pruhs (2003), but we provide a concise proof, for completeness, using our general charging scheme.

Proposition 1 (Kalyanasundaram and Pruhs 2003) *SHORTEST REMAINING PROCESSING TIME FIRST is $2H_k$ -competitive, where H_k denotes the k -th harmonic number, $1 + 1/2 + 1/3 + \cdots + 1/k$.*

Proof We use our general charging scheme. The algorithm is $\frac{k-1}{k}$ -monotone and valid w.r.t. $\pi(i, a) = 1/a$. Observe that whenever the algorithm schedules some job i at time t , then some job will complete in $[t + 1, t + k + 1]$, either i itself or some job with smaller processing time. In particular if t_0 is the completion time of some job i_0 by the algorithm, and s is the smallest time such that $[s, t_0)$ contains no idle time nor completion, then $t_0 - s < k$ and the unit scheduled at time $t_0 - i$ for $1 \leq i \leq t_0 - s + 1$ has capacity at most $1/i$. As a result the total type 2 charge to i_0 is at most H_k .

Lemma 2 states that there are at most $p - 1$ type 3 charges to i_0 from jobs units j with $p_j \leq p$. The worst case is when there is exactly one job unit j with $p_j = p$ charging $1/p$ to i_0 for every $p = 2, 3, \dots, k$. Therefore the total type 3 charge to i_0 is at most $H_k - 1$.

The total type 1 charge is at most $w_{i_0} = 1$, so this concludes the proof. \square

3.2 Lower bound

We start by proving that the analysis of SHORTEST REMAINING PROCESSING TIME FIRST is tight up to a constant factor.

Proposition 2 *The competitive ratio of SHORTEST REMAINING PROCESSING TIME FIRST is at least $\lfloor \log_3(2k + 1) \rfloor$.*

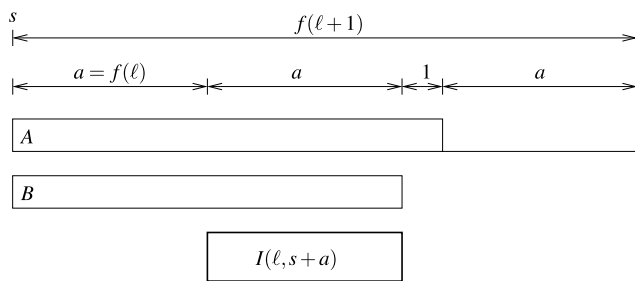


Fig. 2 The construction of $I(\ell + 1, s)$

Proof We define an instance denoted $I(\ell, 0)$ from which the algorithm can complete at most a single job, and the adversary can complete ℓ jobs. Moreover all jobs have processing time at most $(3^\ell - 1)/2$. So if we choose $\ell = \lfloor \log_3(2k + 1) \rfloor$, the processing time is at most k .

Let $\ell \geq 1, s$ be integers. Let f be a function defined as $f(1) = 1$ and for $\ell > 1$,

$$f(\ell) = 3f(\ell - 1) + 1.$$

It can be easily verified by induction that

$$f(\ell) = (3^\ell - 1)/2.$$

We construct recursively instances $I(\ell, s)$ such that in the instance $I(\ell, s)$

1. the adversary can schedule ℓ jobs from this instance,
 2. the algorithm can schedule at most one job from this instance,
 3. all jobs i from the instance satisfy $s \leq r_i < d_i \leq s + f(\ell)$.
- In particular, this implies $p_i \leq f(\ell)$.

For the base case $I(1, s)$, at time s we release a tight job of length 1. It satisfies the required properties.

Now we show how to construct $I(\ell + 1, s)$. Let $a = f(\ell)$. At time s we release a job A of length $2a + 1$ and deadline $s + 3a + 1$, as well as a job B of length $2a$ and tight deadline. Note that SHORTEST REMAINING PROCESSING TIME FIRST will start processing B . At time $s + a$ we release instance $I(\ell, s + a)$; see Fig. 2.

Let us verify that the construction satisfies the required properties, by induction on ℓ . We already settled the base case $\ell = 1$, so assume the claim holds for instances $I(\ell, s')$ for all $s' \geq 0$, and we will show it holds for $I(\ell + 1, s)$ as well. By construction and induction each job i from instance $I(\ell + 1, s)$ is not released before s and its deadline does not exceed $s + 3a + 1 = s + f(\ell + 1)$, so the third property is satisfied.

Until time $s + a$ the algorithm has been processing B for a steps, which means that at that point both A and B are tight for it. Therefore the second property holds, as the algorithm can complete only one job: either A , B , or, by inductive assumption, at most one job from $I(\ell, s + a)$.

On the other hand, the adversary is scheduling the job A from time s to $s + a$. At that point the adversary stops processing A to complete ℓ jobs from $I(\ell, s + a)$ —this is possible by inductive assumption. Afterwards the adversary finishes processing A . Thus the first property holds as well. \square

Next we generalize the former construction, proving a slightly smaller lower bound for every deterministic algorithm. The construction is similar to one known before by Baruah et al. (1994), but we take care to make all the jobs' parameters integral.

Proposition 3 Any deterministic online algorithm has ratio $\Omega(\log k / \log \log k)$.

Proof Fix some deterministic algorithm. We will define an instance denoted $I(\ell, 0, 0)$ from which the algorithm can complete at most a single job, and the adversary can complete ℓ jobs. Moreover all jobs have processing time at most $(\ell + 1)!$. So if we choose $\ell = \lfloor \ln k / \ln \ln k \rfloor - 1$, the processing time is at most

$$\begin{aligned} (\ell + 1)! &= \left\lfloor \frac{\ln k}{\ln \ln k} \right\rfloor! \leq \left(\frac{\ln k}{\ln \ln k} \right)^{\frac{\ln k}{\ln \ln k}} \\ &= \exp \left((\ln \ln k - \ln \ln \ln k) \cdot \frac{\ln k}{\ln \ln k} \right) \\ &\leq \exp(\ln k) = k. \end{aligned}$$

Let $\ell \geq 1, s, e \geq 0$ be integers. Let f be a function defined as $f(1, e) = e + 1$ and for $\ell > 1$,

$$\begin{aligned} f(\ell, e) &= \max\{e, f(\ell - 1, 0)\} + f(\ell - 1, 0) \\ &\quad + f(\ell - 1, \max\{e, f(\ell - 1, 0)\}). \end{aligned} \quad (1)$$

We construct an instance $I(\ell, s, e)$ such that

- the adversary can schedule ℓ jobs from this instance,
- the algorithm can schedule at most one job from this instance, and if it does, then it spends more than e units on jobs from this instance, including uncompleted ones,
- all jobs i from the instance satisfy $r_i \geq s$ and $d_i \leq s + f(\ell, e)$, and therefore also $p_i \leq f(\ell, e)$.

The base case is easy, for $I(1, s, e)$ at time s we release a tight job of length $e + 1$. It satisfies the required properties.

Now we show how to construct $I(\ell + 1, s, e)$. Let $b = f(\ell, 0)$, $a = \max\{e, b\}$ and $c = f(\ell, a)$. At time s we release a job A of length $a + c$ and deadline $s + a + b + c$, as well as a job B of length $a + b$ and tight deadline. At time $s + a$, if the algorithm scheduled only B in $[s, s + a)$, then we release instance $I(\ell, s + a, 0)$. Otherwise at time $s + a + b$ we release $I(\ell, s + a + b, a)$, see Fig. 3.

Let us verify that the construction satisfies the required properties, by induction on ℓ . We already settled the

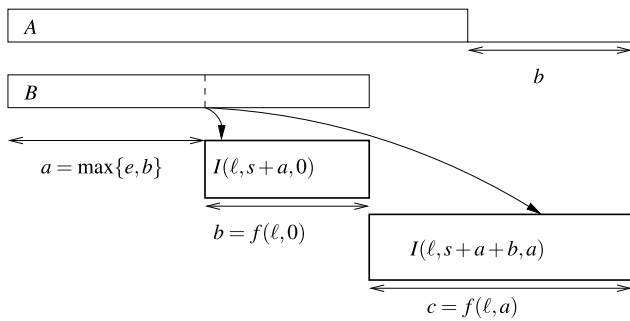


Fig. 3 The construction of $I(\ell + 1, s, e)$

base case $\ell = 1$, so assume the claim holds for instances $I(\ell, s', e')$ for all $s', e' \geq 0$, and we will show it holds for $I(\ell + 1, s, e)$ as well. By construction and induction each job i from instance $I(\ell + 1, s, e)$ is not released before s and its deadline does not exceed $s + a + b + c = s + f(\ell + 1, e)$, so the third property is satisfied.

Case 1 (The algorithm scheduled only B in $[s, s + a]$) At this point, if the algorithm completes A or B , then in the interval $[s + a, s + a + b]$ there is not a single idle step left for another job. Therefore by induction hypothesis the algorithm can only schedule a single job. The algorithm already spent a units on B , so if it does complete a job, then it spends more than $a \geq e$ units on jobs from this instance. By induction hypothesis, the adversary can schedule ℓ jobs from the subinstance in the interval $[s + a, s + a + b]$, and schedule A in the remaining time units $[s, s + a) \cup [s + a + b, s + a + c)$.

Case 2 (The opposite case) The algorithm cannot complete B , since the job is tight. If the algorithm completes some job from $I(\ell, s + a + b, a)$, then by induction hypothesis, it spends strictly more than $a \geq e$ units on jobs from the sub-instance. This does not leave enough space to complete job A in addition. And if the algorithm completes job A , it spends $a + c > e$ units on it. The adversary can complete B plus ℓ jobs from the subinstance.

To complete the proof of the proposition, it remains to show that all jobs from $I(\ell, 0, 0)$ have processing time at most $(\ell + 1)!$. To this end, we prove by induction that

$$f(\ell, e) = \ell \max\{\ell!, (\ell - 1)! + e\}, \quad (2)$$

which implies that all jobs from $I(\ell, 0, 0)$ have processing time at most $\ell \cdot \ell! < (\ell + 1)!$. Note that (2) trivially holds for $\ell = 1$. Now assume it holds for $\ell - 1$, and in particular $f(\ell - 1, 0) = (\ell - 1)(\ell - 1)!$. Then

$$\begin{aligned} f(\ell, e) &= \max\{e, f(\ell - 1, 0)\} + f(\ell - 1, 0) \\ &\quad + f(\ell - 1, \max\{e, f(\ell - 1, 0)\}) \\ &= \max\{e, (\ell - 1) \cdot (\ell - 1)!\} + (\ell - 1) \cdot (\ell - 1)! \end{aligned}$$

$$\begin{aligned} &+ (\ell - 1) \max\{(\ell - 1)!, (\ell - 2)!\} \\ &+ \max\{e, (\ell - 1) \cdot (\ell - 1)!\} \\ &= \max\{e, (\ell - 1) \cdot (\ell - 1)!\} + (\ell - 1) \cdot (\ell - 1)! \\ &\quad + (\ell - 1) \cdot ((\ell - 2)!) \\ &\quad + \max\{e, (\ell - 1) \cdot (\ell - 1)!\} \\ &= \ell \cdot \max\{e, (\ell - 1) \cdot (\ell - 1)!\} + \ell! \\ &= \ell \cdot \max\{e + (\ell - 1)!, \ell!\}. \end{aligned} \quad (3)$$

The equality (3) follows from

$$(\ell - 1)! < (\ell - 2)! + \max\{e, (\ell - 1) \cdot (\ell - 1)!\}. \quad \square$$

4 Bounded processing times, arbitrary weights

4.1 Upper bound

This time we consider instances with arbitrary weights. A natural algorithm for this model, the SMITH RATIO ALGORITHM, schedules the pending job j that maximizes the Smith ratio w_j/p_j at every step. A very simple instance with only two jobs ($r_a = r_b = 0$, $p_a = d_a = w_a = k$, $p_b = 1$, $w_b = 1 + \varepsilon$, $d_b = k + 1$) shows that its competitive ratio is no better than $k + 1$. It turns out that $2k$ -competitiveness can be proved just as easily using our charging scheme. We give the proof for completeness, and then introduce an optimal algorithm.

Proposition 4 *The SMITH RATIO ALGORITHM is $2k$ -competitive.*

Proof We use the general charging scheme. The algorithm is $\frac{k-1}{k}$ -monotone and valid w.r.t. $\pi(i, a) = w_i/a$. Each job i_0 completed by the algorithm receives at most w_{i_0} type 1 charge in total. Lemma 1 implies that each i_0 receives at most kw_{i_0} type 2 charges in total, as for $\pi(i, a) = w_i/a$ the value of ρ is $1 - 1/k$. By Lemma 2, i_0 receives at most $k - 1$ type 3 charges, and each such charge is at most $\pi(i_0, 1) = w_{i_0}$. This concludes the proof. \square

THE EXPONENTIAL CAPACITY ALGORITHM in every step schedules the job j that maximizes $\pi(j, q_j) = w_j \cdot \alpha^{q_j-1}$, which is the capacity function we use in the analysis; $\alpha < 1$ is a parameter that we specify later.

In fact, the constant α depends on k , seemingly making EXPONENTIAL CAPACITY ALGORITHM semi-online. However, the $\alpha(k)$ we use is an increasing function of k , and the algorithm can be made fully online by using the value $\alpha(k^*)$ in each step, where k^* is the maximum processing time among all jobs released up to that step. Let π^* denote the capacity function defined by $\alpha(k^*)$. As π^* only

increases as time goes on, it is straightforward to observe that the algorithm can be analyzed using the final values of k^* and π^* .

Theorem 1 *The EXPONENTIAL CAPACITY ALGORITHM is $(3 + o(1))k/\ln k$ -competitive.*

Proof As before, we use the general charging scheme. Let us define the proper value of $\alpha(k)$ now: $\alpha(k) = 1 - c^2 \cdot \ln k/k$, where $c = 1 - \varepsilon$ for arbitrarily small $\varepsilon > 0$. The algorithm is clearly α -monotone.

To prove validity it suffices to prove that $p\alpha^{p-1} \geq 1$ for all $p \leq k$, as this implies $w_j/p_j \leq w_j\alpha^{p_j-1} \leq \pi(j, q_j)$. Note that $\pi(j, q_j) = w_j/f(q_j)$, where $f(x) \equiv x\alpha^{x-1}$. Thus to prove validity it is enough to exhibit that $f(x) \geq 1$ for $x \in \{1, 2, \dots, k\}$. This holds because of the following claim, which we prove at the end of this proof.

$$f(x) \geq 1 \quad \text{for } 1 \leq x \leq \frac{k}{c^2 \ln k}, \quad (4)$$

$$f(x) \geq \ln k \quad \text{for } \frac{k}{c^2 \ln k} < x \leq k. \quad (5)$$

Now we turn to bounding the charges any job can receive. Suppose that the algorithm schedules job h at time t , and let j be any job pending at t . Let i_0 be the first job completed by the algorithm from time $t + 1$ on. Then, by (4) and (5), the algorithm's monotonicity, and the choice of π ,

$$\begin{aligned} w_j/p_j &\leq w_j\alpha^{p_j-1} \leq \pi(j, q_j(t)) \leq \pi(h, q_h(t)) \\ &\leq \pi(i_0, 1) = w_{i_0}. \end{aligned} \quad (6)$$

Now we bound the total charge of type 3 any job i_0 can receive. Let \mathcal{J} denote the set of job units that are type 3 charged to i_0 . For each $(j, b) \in \mathcal{J}$ the charge from it is w_j/p_j , while $w_j\alpha^{p_j-1} \leq w_{i_0}$, by (6). Therefore, $w_j/p_j \leq w_{i_0}/(p_j\alpha^{p_j-1}) = w_{i_0}/f(p_j)$. Recall that for every $p \leq k$ the number of $(j, b) \in \mathcal{J}$ such that $p_j \leq p$ is at most $p - 1$ by Lemma 2. Applying this for $p = k/(c^2 \ln k)$ and $p = k$, as well as using (4) and (5), we get

$$\sum_{(j,b) \in \mathcal{J}} 1/f(p_j) \leq \frac{k}{c^2 \ln k} + \frac{k}{\ln k} = \frac{k}{\ln k} \left(1 + \frac{1}{c^2}\right).$$

Putting things together, each job i_0 completed by the algorithm receives a type 1 charge of at most w_{i_0} . By Lemma 1 for $\rho = \alpha$, it can receive at most $w_{i_0}k/c^2 \ln k$ type 2 charges in total. And we have just shown that type 3 charges are, for large k , at most $w_{i_0}(1 + 1/c^2)k/\ln k$ in total. Together, this is

$$w_{i_0}((1 + 2/c^2) \cdot k/\ln k + 1) = w_{i_0}(3 + o(1)) \cdot k/\ln k.$$

It remains to prove the claims (4) and (5). To this end let us first observe that for every constant $c < 1$ and large

enough x ,

$$\left(1 - \frac{c}{x}\right)^x \geq \frac{1}{e}, \quad (7)$$

as for x tending to infinity the left-hand side tends to $e^{-c} > e^{-1}$.

Clearly $f(1) = 1$, and if k is sufficiently large, then, by (7),

$$\begin{aligned} f(k) &= k \left(1 - \frac{c^2 \ln k}{k}\right)^{k-1} \\ &= k \left(1 - \frac{c^2 \ln k}{k}\right)^{\frac{k}{c \ln k} (k-1) \frac{c \ln k}{k}} \\ &\geq k \left(\frac{1}{e}\right)^{(k-1) \frac{c \ln k}{k}} \\ &= k \cdot k^{c(1-k)/k} = k^{(1-\varepsilon+k\varepsilon)/k} \\ &\geq \ln k. \end{aligned}$$

Now we observe that the sequence $(f(x))_{x=1}^k$ is non-decreasing for $x \leq k/(c^2 \ln k)$ and decreasing for $x > k/(c^2 \ln k)$. For this we analyze the ratio $f(x)/f(x-1) = \alpha x/(x-1)$, and see that it is at least 1 if and only if $x \leq k/(c^2 \ln k)$. Inequalities (4) and (5) follow. This completes the proof. \square

4.2 Lower bound

Ting (2008) showed that the competitive ratio of any deterministic algorithm in this setting is at least $k/(2 \ln k) - 1$. We improve this bound to $k/\ln k - o(1)$.

Lemma 3 *For any deterministic algorithm the competitive ratio is at least $k/\ln k - o(1)$. In particular, it is at least $k/\ln k - 0.06$ for $k \geq 16$.*

Proof For convenience denote $R = k/\ln k$, $r = \lceil R \rceil - 1$, and assume $k \geq 16$. Fix any deterministic algorithm and consider the following instance, depicted in Fig. 4. At time 0, the adversary releases a big job B with weight $w_B = R$, processing time k , and deadline k , as well as a small job A_1 with weight, processing time, and deadline all 1. Moreover, at each moment $0 \leq t \leq k - 1$, if the algorithm scheduled only job B in $[0, t)$, then the adversary releases a tight job A_{t+1} of unit processing time at time t , and does not release any new job otherwise. The jobs A_t have weights:

$$w(A_t) := \begin{cases} 1 & \text{if } t < R, \\ e^{t/R-1} & \text{if } t \geq R. \end{cases}$$

Note that job A_t is released at time $t - 1$.

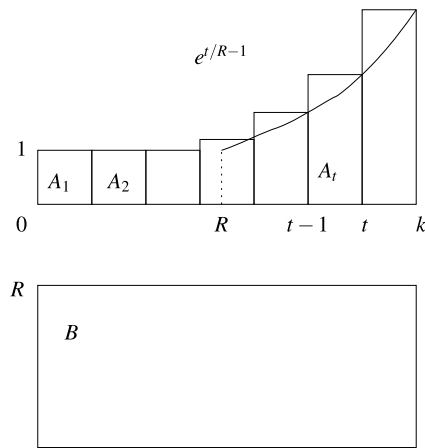


Fig. 4 The construction of the lower bound

If the algorithm schedules a job A_{t_0} with $t_0 < R$, then the adversary schedules job B and the ratio is R .

If the algorithm schedules a job A_{t_0} with $t_0 \geq R$, then the adversary schedules all jobs A_t for $t = 1, \dots, t_0$. The adversary's gain is

$$\begin{aligned} \lceil R \rceil - 1 + \sum_{t=\lceil R \rceil}^{t_0} e^{t/R-1} &= r + \sum_{t=r+1}^{t_0} e^{t/R-1} \\ &\geq r + \int_r^{t_0} e^{t/R-1} dt \\ &= r + [Re^{t/R-1}]_r^{t_0} \\ &= r - Re^{r/R-1} + Re^{t_0/R-1}, \end{aligned} \quad (8)$$

where the inequality follows from monotonicity of the function $e^{t/R-1}$.

We introduce a function f defined as follows:

$$f(R, r) \equiv r - Re^{r/R-1},$$

and rewrite (8) as

$$\begin{aligned} \lceil R \rceil - 1 + \sum_{t=\lceil R \rceil}^{t_0} e^{t/R-1} &\geq f(R, r) + Re^{t_0/R-1} \\ &= f(R, r) + Rw(A_{t_0}). \end{aligned} \quad (9)$$

So the adversary gain is at least $k/\ln k$ times the algorithm's gain plus $f(R, r)$. Later we will prove that while $f(R, r)$ is negative, it tends to 0 as k grows.

If the algorithm schedules job B , gaining $k/\ln k$, the adversary schedules all k jobs A_t from $t = 0$ to $k - 1$. In that case, by (9), its gain is at least

$$\begin{aligned} f(R, r) + Re^{k/R-1} &= f(R, r) + Re^{\ln k-1} \\ &= f(R, r) + R \cdot k/e, \end{aligned}$$

and we need it to be more than $f(R, r) + Rw(B) = f(R, r) + R^2$. This is true if $e \leq \ln k$ which holds for $k \geq e^e$, in particular when $k \geq 16$.

Now we analyze the function $f(R, r)$. Recall that $R = k/\ln k$ and $r = \lceil R \rceil - 1$, so in particular $R - r \in (0, 1]$. As $e^x \geq 1 + x$ and both sides converge to 1 as x tends to 0, we have

$$f(R, r) = r - Re^{r/R-1} \leq r - R \cdot \frac{r}{R} = 0,$$

and $f(R, r)$ tends to 0 as k grows.

We note that f is decreasing in R and increasing in r . Hence we introduce another function

$$g(R) \equiv f(R, R - 1),$$

and note that g is increasing in R , and that for all k ,

$$f(R, r) \geq g(R).$$

Therefore for $k > 21$,

$$f(R, r) \geq g(8) > -0.06,$$

while for $k = 16, 17, \dots, 21$ one can easily check that

$$f(R, r) \geq -0.06$$

as well.

As the algorithm's gain is (w.l.o.g.) at least 1, $f(R, r)$ divided by that gain is at least $f(R, r)$, which concludes the proof. \square

5 Almost identical processing times

In this section we consider instances where each job has an arbitrary weight, and a processing time no less than l and no greater than k . We develop a constant-competitive deterministic algorithm for the case when l/k is a constant. In particular, that algorithm is 5-competitive in the case when $l = k$, for which we also give a lower bound of $3\sqrt{3}/2$.

5.1 Upper bound

THE CONSERVATIVE ALGORITHM: At every step execute the pending job which maximizes the capacity $\pi(j, q_j) = x^{-q_j/p_j} \cdot w_j$. The parameter $x > 1$ may depend on l/k .

Theorem 2 *The CONSERVATIVE ALGORITHM with parameter x is $(1 + x/(1 - x^{-l/k}))$ -competitive, where k and l denote, respectively, the maximum and minimum processing time of the jobs.*

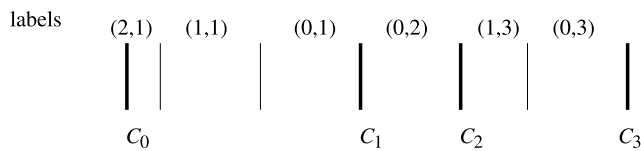


Fig. 5 The (sub)intervals as used by the charging procedure

Proof The proof is based on a charging scheme, slightly different from the general charging scheme of Sect. 2. It can still be viewed as a variant of that charging scheme, in which whole jobs are charged from rather than individual job units, and special care is taken to prevent charges of type 3.

Fix some instance. Consider the jobs scheduled by the algorithm and jobs scheduled by the adversary. Without loss of generality we assume that the adversary completes every job that he starts, and that he completes them in the EARLIEST DEADLINE FIRST order. To simplify the argument, we assume that ties between deadlines are broken in an arbitrary way.

We also assume without loss of generality that whenever the algorithm has no pending jobs at the very beginning of some step, the adversary will release no further jobs until he has no pending jobs as well for at least one step. This partitions the sequence into independent phases in a natural way. From now on we analyze a single phase.

Every job j scheduled by the adversary, that is also completed by the algorithm, is charged to itself. From now on we ignore such jobs, and focus on the remaining ones.

All jobs scheduled by the adversary will be charged to some jobs completed by the algorithm, in such a way that job i completed by the algorithm receives a charge of at most $w_i x / (1 - x^{-1/k})$ in total.

For convenience we renumber the jobs completed by the algorithm from 1 to n , such that the completion times are ordered $C_1 < \dots < C_n$. Also, we let $C_0 = 0$. For every $i = 1, \dots, n$ we divide $[C_{i-1}, C_i]$ further into subintervals. Let $a_i = \lceil (C_i - C_{i-1})/l \rceil$. The first subinterval is $[C_{i-1}, C_i - (a_i - 1)l]$. The remaining subintervals are $[C_i - (b+1)l, C_i - bl]$ for every $b = a_i - 2, \dots, 0$. We label every subinterval I with a pair (b, i) such that $I = [s, C_i - bl]$ for $s = \max\{C_{i-1}, C_i - (b+1)l\}$. The division of intervals and their numbering is illustrated in Fig. 5.

The charging is done by the following procedure, which maintains for every interval $[s, t)$ a set of jobs P that are started before t by the adversary and that are not yet charged to any job of the algorithm.

Initially $P = \emptyset$.

For all subintervals $[s, t)$ as defined above in left to right order, do

- Let (b, i) be the label of the subinterval.
- Add to P all jobs started by the adversary in $[s, t)$.

- If P is not empty, then remove from P the job j with the smallest deadline and charge it to i . Mark $[s, t)$ with j .
- If P is empty, then leave $[s, t)$ unmarked.
- Denote by P_t the current content of P .

Lemma 4 For every subinterval $[s, t)$, all jobs $j \in P_t$ are still pending for the algorithm at time t .

Proof Assume that P_t is not empty, and let j be the earliest-deadline job in P_t .

First we claim that there is a time $s_0 \leq r_j$ such that every subinterval contained in $[s_0, t)$ is marked with some job j' such that $r_{j'} \geq s_0$ and $d_{j'} < d_j$.

Indeed, let s_0 be the minimal starting point of any subinterval in this phase such that all the subintervals contained in $[s_0, t)$ are marked with some job j' satisfying $d_{j'} < d_j$. Note that s_0 exists since t is a candidate.

Suppose that s_0 is the beginning of the phase or the subinterval that ends at s_0 is unmarked. Then j and all the jobs j' that mark the subintervals contained in $[s_0, t)$ were not released before s_0 .

In the opposite case, the subinterval that ends at s_0 is marked with a job j'' such that $d_{j''} > d_j$. Thus $r_j \geq s_0$ and all the jobs j' marking the subintervals contained in $[s_0, t)$ also satisfy $r_{j'} \geq s_0$, since otherwise one of them would be selected instead of j'' by the marking procedure before. That proves the claim.

Now let \mathcal{M} be the set of jobs charged during all subintervals in $[s_0, t)$. We claim that the adversary completes all jobs in \mathcal{M} before he completes j . To prove this claim, we start by renumbering the subintervals in $[s_0, t)$ with $1, 2, \dots, n$. Each subinterval has length at most l , and each job in \mathcal{M} has processing time at least l . And by the marking procedure, for all $m \leq n$ the adversary starts at least m jobs from \mathcal{M} in the first m subintervals. Thus, since jobs in \mathcal{M} have deadlines smaller than d_j by the previous claim, the adversary completes them all before j due to the EARLIEST DEADLINE FIRST order.

Since the adversary completes j , its deadline satisfies $d_j \geq s_0 + |\mathcal{M}|l + p_j$. On the other hand, as the subintervals have length at most l , we have $t \leq s_0 + |\mathcal{M}|l$. Therefore $d_j - t \geq p_j$, which shows that j is still pending for the algorithm at t . \square

Lemma 5 Let $[s, t)$ be a subinterval with label (b, i) and j a job pending for the algorithm at some time $t_0 \in [s, t)$. Then $w_j \leq x^{1-bl/k} w_i$.

Proof Let $u = C_i$ and let $y_{t_0}, y_{t_0+1}, \dots, y_{u-1}$ be the respective priorities of the job units scheduled in $[t_0, u)$. Clearly, the algorithm is $x^{-1/k}$ -monotone, i.e., $y_{t'} \leq x^{-1/k} y_{t'+1}$ for every $t' \in [t_0, u)$.

Note that $y_{u-1} = x^{-1/p_i} w_i \leq x^{-1/k} w_i$ since i completes at u . As the capacity of j at time t_0 is at least $x^{-1} w_j$ and $u - t_0 \geq bl$,

$$\begin{aligned} x^{-1} w_j &\leq \pi(j, q_j(t_0)) \leq y_{t_0} \leq x^{-(u-1-t_0)/k} y_{u-1} \\ &\leq x^{-(u-t_0)/k} w_i = x^{-(u-t_0)/l \cdot l/k} w_i \leq x^{-bl/k} w_i, \end{aligned}$$

where the third inequality is due to the $x^{-1/k}$ -monotonicity of the algorithm. \square

This lemma permits us to bound the total charge of a job i completed by the algorithm. Let $a = \lceil (C_i - C_{i-1})/l \rceil$. Then i gets at most one charge of weight at most $x^{1-bl/k} w_i$ for every $b = a - 1, \dots, 0$. Summing the bounds shows that the charge that job i receives is bounded by

$$w_i \sum_{b=0}^{\infty} x^{1-bl/k} = \frac{w_i x}{1 - x^{-l/k}},$$

plus one possible self-charge of weight w_i .

At time $t = C_n$ the algorithm is idle, so $P_t = \emptyset$ by Lemma 4. Therefore all jobs scheduled by the adversary have been charged to some job of the algorithm, and this completes the proof. \square

It is straightforward to check that for a given l/k ratio, the value of x that minimizes our upper bound on the competitive ratio of CONSERVATIVE ALGORITHM is $x = (1 + l/k)^{k/l}$. Using that value yields ratio $1 + (1 + k/l)(1 + l/k)^{k/l} \leq 1 + e(1 + k/l)$. When no information on l/k is available in advance, one possible choice of the parameter is $x = e$, which is the limit of the optimum values of x as l/k tends to zero.

For the important special case of identical processing times (i.e., $l = k$), the optimum value of x is 2. It yields the following bound.

Corollary 1 *The CONSERVATIVE ALGORITHM with parameter $x = 2$ is 5-competitive when all the jobs have the same processing time.*

5.2 Lower bound

In this section we present a relevant lower bound, in which all the jobs have exactly the same processing times. Precisely, we analyze the lower bound construction of Chan et al. (2004). They proved it forces ratio 2.59, and we improve this to $\frac{3}{2} \cdot \sqrt{3} \approx 2.598$. We remark that this is the threshold value of competitive ratio that can be forced with that construction.

Theorem 3 *Any deterministic online algorithm for the equal processing time variant with $l = k \geq 2$ has competitive ratio at least $\frac{3}{2} \cdot \sqrt{3} \approx 2.598$.*

Proof We describe the adversary's strategy for $k = 2$ only, as it can be easily adapted to larger values of k . Every job j will be tight, i.e., $d_j = r_j + p_j = r_j + 2$. We specify the set of jobs completed by the adversary once the sequence is finished, and only describe job releases for the time being. We also assume that when there are pending jobs with positive weights, ALG will process one of them, and that it will never process a job with non-positive weight.

Initially ($t = 0$) the adversary releases a job with weight $x_0 = 1$. In every step $t > 0$ the adversary releases a job with weight x_t that we specify later, unless the algorithm has already completed one job (the one with weight x_{t-2}). In that case the adversary releases no job at time t and the sequence is finished. The adversary completes every other job starting from the last one, for a total gain of

$$X_{t-1} = x_{t-1} + x_{t-3} + \dots + x_{b+2} + x_b,$$

where $b = t - 1 \bmod 2$, while ALG's gain is only x_{t-2} .

Now we describe the sequence x_i that forces ratio at least $R = \frac{3}{2}\sqrt{3} - \varepsilon$ for an arbitrarily small $\varepsilon > 0$. As we later prove, there is a non-positive element x_{i_0} in the sequence, so, by previous assumptions, the algorithm completes some job released before the step i_0 .

If ALG completes a job released in step t , the ratio is

$$R_t = \frac{X_{t+1}}{x_t} = \frac{X_{t+1}}{X_t - X_{t-2}},$$

assuming $X_{-2} = X_{-1} = 0$. As we want to force ratio R , we let $R_t = R$, i.e.,

$$X_{t+1} = R(X_t - X_{t-2})$$

for each $t > 0$. Note that this defines the sequence x_i , as $x_i = X_i - X_{i-2}$.

Now we introduce a sequence $\{s_i\}_{i \geq 0}$:

$$s_i \equiv R(1 - X_{i-1}/X_{i+1}).$$

We prove that there exists an i such that $s_i \leq 0$, which implies that there exists an i_0 such that $x_{i_0} \leq 0$.

Assume for contradiction that $s_i > 0$ for all i . Under the assumption we prove by induction that $\{s_i\}_{i \geq 0}$ is a decreasing sequence. Observe that

$$\begin{aligned} X_i &= R(X_{i-1} - X_{i-3}) = X_{i-1} \cdot R \left(1 - \frac{X_{i-3}}{X_{i-1}} \right) \\ &= X_{i-1} \cdot s_{i-2}, \end{aligned}$$

which implies

$$s_i = R \left(1 - \frac{1}{s_{i-1}s_{i-2}} \right). \quad (10)$$

It follows from (10) that $s_0 = R$, $s_1 = R - 1/R$, and $s_2 = R(R^2 - 2)/(R^2 - 1)$. In particular $s_0 > s_1 > s_2 > 0$, which establishes the induction basis.

For the inductive step observe that

$$\begin{aligned} s_{i+1} - s_i &= R \left(\frac{1}{s_{i-1}s_{i-2}} - \frac{1}{s_i s_{i-1}} \right) \\ &= R \cdot \frac{s_i - s_{i-2}}{s_i s_{i-1} s_{i-2}} < 0, \end{aligned}$$

since, by induction hypothesis, $s_{i-2} > s_{i-1} > s_i$.

As the sequence $\{s_i\}_{i \geq 0}$ is positive and decreasing, it converges to $\inf s_i = g \geq 0$. Then, by (10),

$$g = R \left(1 - \frac{1}{g^2} \right),$$

or, equivalently,

$$P(g) = g^3 - Rg^2 + R = 0.$$

Since $R = \frac{3}{2}\sqrt{3} - \varepsilon$, the discriminant of P , $4R^2(R^2 - \frac{27}{4})$, is negative, so P has a single real root. This sole root lies in $(-1, 0)$, as $P(-1) = -1$ and $P(0) = R > 0$. Thus $g < 0$, a contradiction. \square

6 Note on randomized algorithms

As we noted in the introduction, there is a $O(1)$ -competitive randomized online algorithm for the variant of unit weight jobs of arbitrary processing times (Kalyanasundaram and Pruhs 2003), but when the jobs have arbitrary weights and processing times at most k , the competitive ratio becomes $\Omega(\sqrt{\log k / \log \log k})$ (Canetti and Irani 1998). The same paper gives an $O(\log k)$ -competitive randomized algorithm for a more restricted setting with tight jobs only. Using the well-known “classify and randomly select” paradigm (Awerbuch et al. 1994) and our CONSERVATIVE ALGORITHM, we generalize the latter result as follows.

Corollary 2 *Using the CONSERVATIVE ALGORITHM and the “classify and randomly select” paradigm yields a $7.75\lceil \log k \rceil$ -competitive randomized algorithm for the variant with jobs with processing time bounded by k and arbitrary weights and deadlines.*

Proof Given k , the upper bound on processing times, one can proceed as follows. Divide the interval $[1, k]$ into subintervals $I_0, I_1, \dots, I_{\lceil \log k \rceil}$, defined by $I_j \equiv [2^j, 2^{j+1})$. Select one of these subintervals uniformly at random and denote it I^* . Denote the set of jobs with processing times from I^* by J . Use the CONSERVATIVE ALGORITHM for scheduling jobs from J and ignore all other jobs. As the ratio of minimum to maximum processing time in J is at least $1/2$, the

CONSERVATIVE ALGORITHM is 7.75 -competitive with respect to the optimum on J . In turn, the expected optimum on J is at least $1/(1 + \lceil \log k \rceil)$ of the optimum on the whole instance, due to our random choice of I^* . \square

7 Conclusion

It remains open to determine the best competitive ratio a deterministic algorithm can achieve for the equal processing time model. Even for $k = 1$ the question is not completely answered.

Much less is known about randomized algorithms. When one considers jobs with processing time bounded by k and no further restrictions, there is a huge gap between the lower bound of $\Omega(\sqrt{\log k / \log \log k})$ (Canetti and Irani 1998) and the $O(\log k)$ upper bound we demonstrated. We are also not aware of any work on randomized algorithms for the variant of jobs with identical processing times (greater than one) and arbitrary weights and deadlines.

Acknowledgements We would like to thank Artur Jeż for his valuable comments.

Christoph Dürr was supported by Agence Nationale pour la Recherche (ANR) project ALgorithmique des Plates-formes A Grande Echelle (Alpage).

Łukasz Jeż was supported by Ministertwo Nauki i Szkolnictwa Wyższego (MNiSW) grants no. N N206 1723 33 2007–2010 and N N206 490638 2010–2011, and European Cooperation in Science and Technology (COST) 295: Foundations and Algorithms for Dynamic Networks (DYNAMO).

Open Access This article is distributed under the terms of the Creative Commons Attribution Noncommercial License which permits any noncommercial use, distribution, and reproduction in any medium, provided the original author(s) and source are credited.

References

- Awerbuch, B., Bartal, Y., Fiat, A., & Rosén, A. (1994). Competitive non-preemptive call control. In *Proc. 5th symp. on discrete algorithms* (pp. 312–320).
- Baptiste, P. (1999). An $O(n^4)$ algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Operations Research Letters*, 24(4), 175–180.
- Baptiste, P., Chrobak, M., Dürr, C., Jawor, W., & Vakhania, N. (2004). Preemptive scheduling of equal-length jobs to maximize weighted throughput. *Operations Research Letters*, 32(3), 258–264.
- Baruah, S., Haritsa, J., & Sharma, N. (1994). On-line scheduling to maximize task completions. In *Real-time systems symposium* (pp. 228–236).
- Canetti, R., & Irani, S. (1998). Bounding the power of preemption in randomized scheduling. *SIAM Journal on Computing*, 27(4), 993–1015.
- Chan, W. T., Lam, T. W., Ting, H. F., & Wong, P. W. H. (2004). New results on on-demand broadcasting with deadline via job scheduling with cancellation. In *Proc. 10th international computing and combinatorics conference* (pp. 210–218).

- Chen, B., Potts, C. N., & Woeginger, G. J. (1998). A review of machine scheduling: complexity, algorithms and approximability. In *Handbook of combinatorial optimization* (Vol. 3, pp. 21–169). Dordrecht: Kluwer Academic.
- Chrobak, M., Jawor, W., Sgall, J., & Tichý, T. (2007). Online scheduling of equal-length jobs: randomization and restarts help. *SIAM Journal on Computing*, 36(6), 1709–1728.
- Englert, M., & Westermann, M. (2007). Considering suppressed packets improves buffer management in QoS switches. In *Proc. 18th symp. on discrete algorithms (SODA)* (pp. 209–218).
- Hajek, B. (2001). On the competitiveness of online scheduling of unit-length packets with hard deadlines in slotted time. In *Proceedings of conference on information sciences and systems (CISS)* (pp. 434–438).
- Kalyanasundaram, B., & Pruhs, K. (2000). Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4), 617–643.
- Kalyanasundaram, B., & Pruhs, K. (2003). Maximizing job completions online. *Journal of Algorithms*, 49(1), 63–85.
- Lawler, E.L. (1994). Knapsack-like scheduling problems, the Moore-Hodgson algorithm and the “tower of sets” property. *Mathematical and Computer Modelling*, 20(2), 91–106.
- Ting, H. F. (2008). A near optimal scheduler for on-demand data broadcasts. *Theoretical Computer Science*, 401(1–3), 77–84.
- Vakhania, N. (2008). A fast on-line algorithm for the preemptive scheduling of equal-length jobs on a single processor. In *Proceedings of the 2nd WSEAS international conference on computer engineering and applications* (pp. 158–161).